浙江大学

计算机视觉(本科)作业报告

作业名称:	**Bird’s-eye view projection**
姓 名:	曹小川
学 号:	3200105705
电子邮箱:	434446127@qq.com
联系电话:	18991060018
导 师:	潘纲

2022年 2 月 8 日

Bird’s-eye view projection

一、 作业已实现的功能简述及运行简要说明

作业已实现功能：

将图片文件 "OpenCV_Chessboard.png "打印在一张A4纸上作为校准图案。图案，然后把它贴在飞机上。

- 用你的智能手机拍摄近10张不同视角下的图案照片，作为校准输入。
- 用同一部手机拍下你的场景照片（"view.jpg"）。
- 使用图案图像和 "view.jpg "作为输入。
- 输出1：校准结果（相机矩阵和失真系数）。
- 输出2：显示发现的角和每个图案图像的失真校正后的图像。
- 输出3："view.jpg "的鸟瞰图转换。

二、 作业的开发与运行环境

开发工具:Visual Studio

操作系统: Windows 10

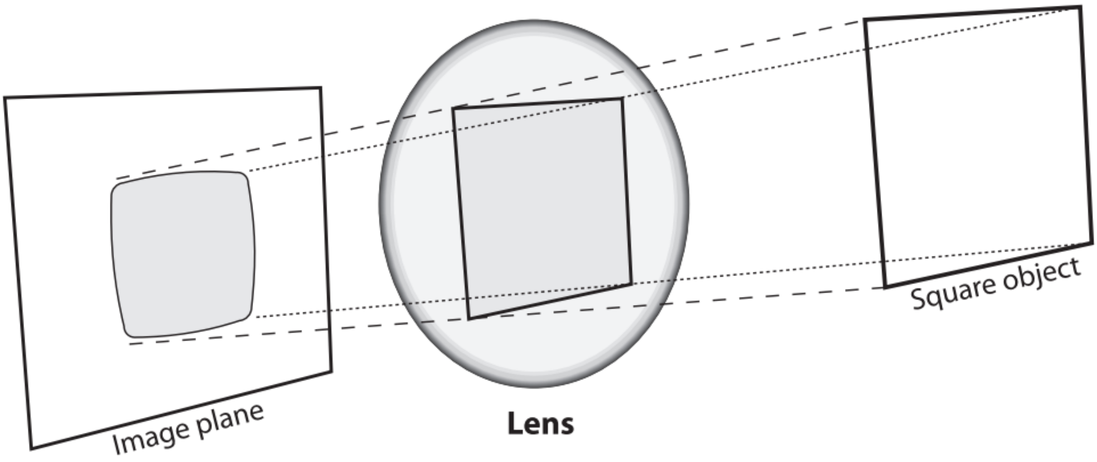
编程语言：C++/ OpenCV

三、 系统或算法的基本思路、原理、及流程或步骤等

将物理世界中坐标为(Xi, Yi, Zi)的点Qi映射到投影屏幕上坐标为(xi, yi)的点的关系，被称为投影变换。在处理这种变换时，使用所谓的同质坐标是很方便的。在维数为n的投影空间中，与一个点相关的同质坐标通常表示为一个（n+1）维的向量（例如，x，y，z变成x，y，z，w），另外还有一个限制，即任何两个数值成比例的点都是等价的。在我们的例子中，图像平面是投影空间，它有两个维度，所以我们将该平面上的点表示为三维向量q=（q1，q2，q3）。回顾所有在投影空间中具有比例值的点都是相等的，我们可以通过除以q3来恢复实际的像素坐标。这样我们就可以把定义相机的参数（即fx、fy、cx和cy）排列成一个3乘3的矩阵，我们称之为相机本征矩阵

$$q = MQ, \quad \text{where} \quad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

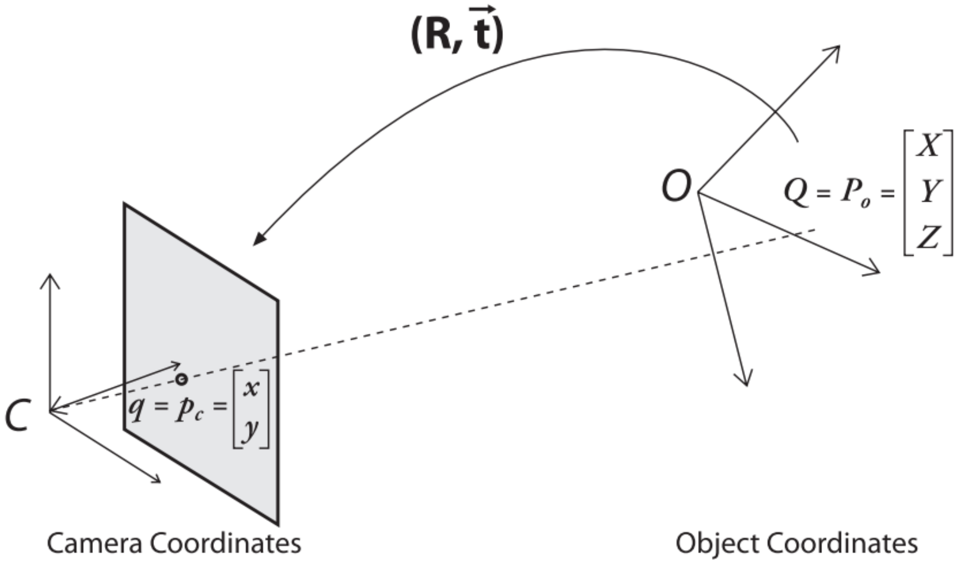
径向畸变：与靠近中心的光线相比，离简单透镜中心较远的光线弯曲得太厉害；因此，正方形的边看起来在图像上弯曲。因此，正方形的边看起来在图像平面上弯曲



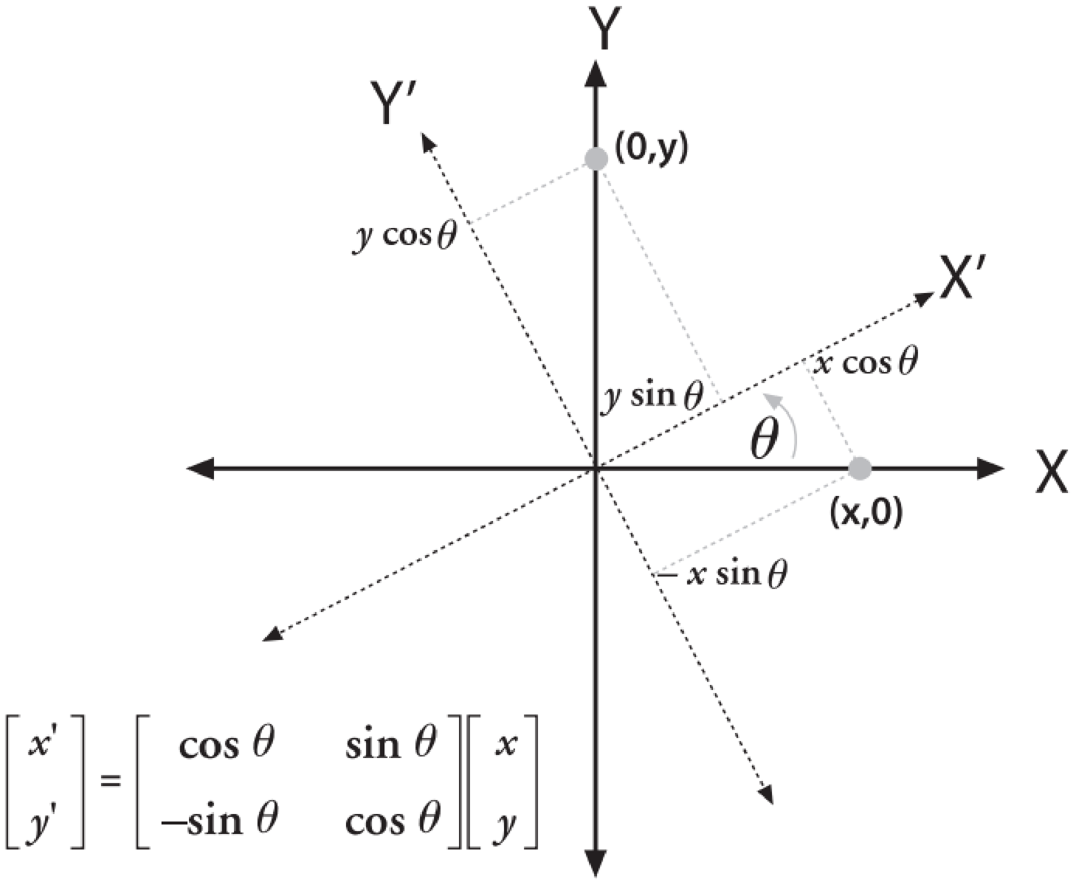
$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

对于相机拍摄的每个特定物体的图像，我们可以用旋转和平移来描述该物体相对于相机坐标系的姿态。

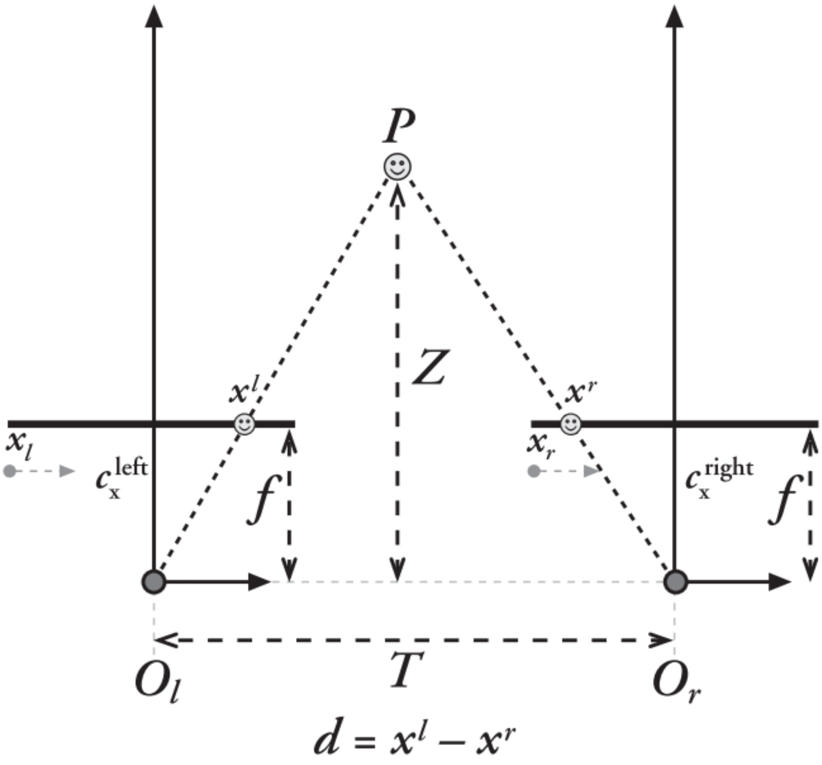

(R, $\vec{t}$)
O
$Q = P_o = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$
C
$q = p_c = \begin{bmatrix} x \\ y \end{bmatrix}$
Camera Coordinates
Object Coordinates

将各点旋转θ（在本例中，围绕Z轴）与将坐标轴反转θ是一样的。坐标轴旋转θ；通过简单的三角函数，我们可以看到旋转如何改变一个点的坐标.



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

在一个完全不失真的、对齐的立体设备和已知的对应关系下，深度Z 可以通过类似的三角形找到；成像器的主射线开始于投影的中心 和Ol 和Or，并通过两个图像平面的主点延伸到cl 和cr.



四、 具体如何实现，例如关键（伪）代码、主要用到函数与算法等

收集所有被发现的板上的角:

```
cv::Size image_size;
int board_count = 0;
```

```
for (size_t i = 0; (i < filenames.size()) && (board_count < n_boards); ++i) {
    cv::Mat image, image0 = cv::imread(folder + filenames[i]);
    board_count += 1;
    if (!image0.data) {  // protect against no file
        cerr << folder + filenames[i] << ", file #" << i << ", is not an image"
<< endl;
        continue;
    }
    image_size = image0.size();
    cv::resize(image0, image, cv::Size(), image_sf, image_sf, cv::INTER_LINEAR);

    vector<cv::Point2f> corners;
    bool found = cv::findChessboardCorners(image, board_sz, corners);
    drawChessboardCorners(image, board_sz, corners, found);
    if (found) {
        image ^= cv::Scalar::all(255);
        cv::Mat mcorners(corners);
        mcorners *= (1.0 / image_sf);
        image_points.push_back(corners);
        object_points.push_back(vector<cv::Point3f>());
        vector<cv::Point3f>& opts = object_points.back();

        opts.resize(board_n);
        for (int j = 0; j < board_n; j++) {
            opts[j] = cv::Point3f(static_cast<float>(j / board_w),
                                  static_cast<float>(j % board_w), 0.0f);
        }
    }
    cv::imshow("Calibration", image);

    if ((cv::waitKey(delay) & 255) == 27) {
        return -1;
    }
}
```

校准相机:

```
cv::Mat intrinsic_matrix, distortion_coeffs;
double err = cv::calibrateCamera(
    object_points, image_points, image_size, intrinsic_matrix,
    distortion_coeffs, cv::noArray(), cv::noArray(),
    cv::CALIB_ZERO_TANGENT_DIST | cv::CALIB_FIX_PRINCIPAL_POINT);
```

保存本征和失真

```
cout << " *** DONE!\n\nReprojection error is " << err
    << "\nStoring Intrinsics.xml and Distortions.xml files\n\n";
cv::FileStorage fs("intrinsics.xml", cv::FileStorage::WRITE);
fs << "image_width" << image_size.width << "image_height" << image_size.height
    << "camera_matrix" << intrinsic_matrix << "distortion_coefficients"
    << distortion_coeffs;
fs.release();
```

得到checkboard

```
vector<cv::Point2f> corners;
bool found = cv::findChessboardCorners( // True if found
    image,                              // Input image
    board_sz,                           // Pattern size
    corners,                            // Results
    cv::CALIB_CB_ADAPTIVE_THRESH | cv::CALIB_CB_FILTER_QUADS);
if (!found) {
    cout << "Couldn't acquire checkerboard on " << argv[7] << ", only found "
        << corners.size() << " of " << board_n << " corners\n";
    return -1;
}
```

找到 homography

```
cv::Mat H = cv::getPerspectiveTransform(objPts, imgPts);
```

显示旋转和平移矩阵

```
vector<cv::Point2f> image_points2;
vector<cv::Point3f> object_points2;
for (int i = 0; i < 4; ++i) {
    image_points2.push_back(imgPts[i]);
    object_points2.push_back(cv::Point3f(objPts[i].x, objPts[i].y, 0));
}
cv::Mat rvec, tvec, rmat;
cv::solvePnP(object_points2,
             image_points2,
             intrinsic,
             cv::Mat(),
             rvec,
             tvec
            );
cv::Rodrigues(rvec, rmat);
```

五、 实验结果与分析

程序代码：

文字识别

```
#undef UNICODE
#include <windows.h>
#include <iostream>
#include <algorithm>
#include <opencv2/opencv.hpp>

using namespace std;

int readFilenames(vector<string>& filenames, const string& directory) {

    WIN32_FIND_DATA data;
    HANDLE dir = FindFirstFile((directory + "/*").c_str(), &data);
    if (dir == INVALID_HANDLE_VALUE)
        return 0;
    do {
        const string file_name = data.cFileName;
        const string full_file_name = directory + "/" + file_name;
        const bool is_directory = (data.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) != 0;
        if (file_name[0] == '.')
            continue;
        if (is_directory)
            continue;
        filenames.push_back(full_file_name);
    } while (FindNextFile(dir, &data));

    FindClose(dir);
    std::sort(filenames.begin(), filenames.end());  //sort the name of files
    return(filenames.size());
}  // GetFilesInDirectory


int main() {
    float image_sf = 0.5f;     // image scaling factor
    int delay = 250;           // miliseconds
    int board_w = 0;
    int board_h = 0;
    cout << "input the width number of board corners:" << endl;
    cin >> board_w;
    cout << "input the height number of board corners:" << endl;
    cin >> board_h;
    int n_boards;
    cout << "input the number of board:" << endl;
    cin >> n_boards;// how many boards max to read
    cout << "input the milisecond delay" << endl;
    cin >> delay;
    cout << "input the image_scaling_factor:" << endl;
    cin >> image_sf;
    string folder;
    cout << "input the name of folder:" << endl;
    cin >> folder;
    string chessB;
    cout << "input path to chessboard" << endl;
    cin >> chessB;

    int board_n = board_w * board_h;  // number of corners
    cv::Size board_sz = cv::Size(board_w, board_h);  // width and height of the
board
    cout << "Reading in directory " << folder << endl;
    vector<string> filenames;
    int num_files = readFilenames(filenames, folder);
    cout << "   ... Done. Number of files = " << num_files << "\n" << endl;

    vector<vector<cv::Point2f> > image_points;
    vector<vector<cv::Point3f> > object_points;
    // collecting all corners on the boards that are found
    cv::Size image_size;
    int board_count = 0;
    for (size_t i = 0; (i < filenames.size()) && (board_count < n_boards); ++i)
{
        cv::Mat image, image0 = cv::imread(folder + filenames[i]);
        board_count += 1;
        if (!image0.data) {  // protect against no file
            cerr << folder + filenames[i] << ", file #" << i << ", is not an
image" << endl;
            continue;
        }
        image_size = image0.size();
        cv::resize(image0, image, cv::Size(), image_sf, image_sf,
cv::INTER_LINEAR);

        vector<cv::Point2f> corners;
        bool found = cv::findChessboardCorners(image, board_sz, corners);

        // Draw
        drawChessboardCorners(image, board_sz, corners, found);  // will draw
only if found
        // If found , add it to our data
        if (found) {
            image ^= cv::Scalar::all(255);
            cv::Mat mcorners(corners);
            // do not copy the data
            mcorners *= (1.0 / image_sf);
            // scale the corner coordinates
            image_points.push_back(corners);
            object_points.push_back(vector<cv::Point3f>());
            vector<cv::Point3f>& opts = object_points.back();

            opts.resize(board_n);
            for (int j = 0; j < board_n; j++) {
                opts[j] = cv::Point3f(static_cast<float>(j / board_w),
                    static_cast<float>(j % board_w), 0.0f);
            }
        }
        cv::imshow("Calibration", image);

        if ((cv::waitKey(delay) & 255) == 27) {
            return -1;
        }
    }

    cv::destroyWindow("Calibration");
    cout << "\n\n*** CALIBRATING THE CAMERA...\n" << endl;

    cv::Mat intrinsic_matrix, distortion_coeffs;
    double err = cv::calibrateCamera(
        object_points, image_points, image_size, intrinsic_matrix,
        distortion_coeffs, cv::noArray(), cv::noArray(),
        cv::CALIB_ZERO_TANGENT_DIST | cv::CALIB_FIX_PRINCIPAL_POINT);
    cout << " *** DONE!\n\nReprojection error is " << err
        << "\nStoring Intrinsics.xml and Distortions.xml files\n\n";
    cv::FileStorage fs("intrinsics.xml", cv::FileStorage::WRITE);
    fs << "image_width" << image_size.width << "image_height" << image_size.height
        << "camera_matrix" << intrinsic_matrix << "distortion_coefficients"
        << distortion_coeffs;
    fs.release();

    fs.open("intrinsics.xml", cv::FileStorage::READ);
    cout << "\nimage width: " << static_cast<int>(fs["image_width"]);
    cout << "\nimage height: " << static_cast<int>(fs["image_height"]);
    cv::Mat intrinsic_matrix_loaded, distortion_coeffs_loaded;
    fs["camera_matrix"] >> intrinsic_matrix_loaded;
    fs["distortion_coefficients"] >> distortion_coeffs_loaded;
    cout << "\nintrinsic matrix:" << intrinsic_matrix_loaded;
    cout << "\ndistortion coefficients: " << distortion_coeffs_loaded << "\n" << endl;

    // Build the undistort map
    cv::Mat map1, map2;
    cv::initUndistortRectifyMap(intrinsic_matrix_loaded, distortion_coeffs_loaded,
        cv::Mat(), intrinsic_matrix_loaded, image_size,
        CV_16SC2, map1, map2);

    //display
    board_count = 0;  // resent max boards to read
    for (size_t i = 0; (i < filenames.size()) && (board_count < n_boards); ++i) {
        cv::Mat image, image0 = cv::imread(folder + filenames[i]);
        ++board_count;
        if (!image0.data) {  // protect against no file
            cerr << folder + filenames[i] << ", file #" << i << ", is not an image" << endl;
            continue;
        }
        cv::remap(image0, image, map1, map2, cv::INTER_LINEAR,
            cv::BORDER_CONSTANT, cv::Scalar());
        cv::imshow("Original", image0);
        cv::imshow("Undistorted", image);
        if ((cv::waitKey(0) & 255) == 27) {
            break;
        }
    }
    cv::destroyWindow("Original");
    cv::destroyWindow("Undistorted");

    //birdview
    cv::FileStorage fs2("intrinsics.xml", cv::FileStorage::READ);
    cv::Mat intrinsic, distortion;
    fs2["camera_matrix"] >> intrinsic;
    fs2["distortion_coefficients"] >> distortion;
    if (!fs2.isOpened() || intrinsic.empty() || distortion.empty()) {
        cout << "Error: Couldn't load intrinsic parameters" << endl;
        return -1;
    }
    fs2.release();

    cv::Mat gray_image, image, image0 = cv::imread(chessB, 1);
    if (image0.empty()) {
        cout << "Error: Couldn't load image " << chessB << endl;
        return -1;
    }
    cv::undistort(image0, image, intrinsic, distortion, intrinsic);
    cv::cvtColor(image, gray_image, cv::COLOR_BGRA2GRAY);

    vector<cv::Point2f> corners;
    bool found = cv::findChessboardCorners( // True if found
        image,                              // Input image
        board_sz,                           // Pattern size
        corners,                            // Results
        cv::CALIB_CB_ADAPTIVE_THRESH | cv::CALIB_CB_FILTER_QUADS);
    if (!found) {
        cout << "Couldn't acquire checkerboard on " << chessB << ", only found "
            << corners.size() << " of " << board_n << " corners\n";
        return -1;
    }

    // Get Subpixel accuracy on those corners
    cv::cornerSubPix(
        gray_image,       // Input image
        corners,          // Initial guesses, also output
        cv::Size(11, 11), // Search window size
        cv::Size(-1, -1), // Zero zone (in this case, don't use)
        cv::TermCriteria(cv::TermCriteria::EPS | cv::TermCriteria::COUNT, 30,
            0.1));

    cv::Point2f objPts[4], imgPts[4];
    objPts[0].x = 0;
    objPts[0].y = 0;
    objPts[1].x = board_w - 1;
    objPts[1].y = 0;
    objPts[2].x = 0;
    objPts[2].y = board_h - 1;
    objPts[3].x = board_w - 1;
    objPts[3].y = board_h - 1;
    imgPts[0] = corners[0];
    imgPts[1] = corners[board_w - 1];
    imgPts[2] = corners[(board_h - 1) * board_w];
    imgPts[3] = corners[(board_h - 1) * board_w + board_w - 1];

    cv::circle(image, imgPts[0], 9, cv::Scalar(255, 0, 0), 3);
    cv::circle(image, imgPts[1], 9, cv::Scalar(0, 255, 0), 3);
    cv::circle(image, imgPts[2], 9, cv::Scalar(0, 0, 255), 3);
    cv::circle(image, imgPts[3], 9, cv::Scalar(0, 255, 255), 3);

    cv::drawChessboardCorners(image, board_sz, corners, found);
    cv::imshow("Checkers", image);
    cv::Mat H = cv::getPerspectiveTransform(objPts, imgPts);

    cout << "\nPress 'd' for lower birdseye view, and 'u' for higher (it adjusts the apparent 'Z' height), Esc to exit" << endl;
    double Z = 15;
    cv::Mat birds_image;
    for (;;) {
        // escape key stops
        H.at<double>(2, 2) = Z;

        cv::warpPerspective(image,          // Source image
            birds_image,    // Output image
            H,              // Transformation matrix
            image.size(),   // Size for output image
            cv::WARP_INVERSE_MAP | cv::INTER_LINEAR,
            cv::BORDER_CONSTANT, cv::Scalar::all(0) // Fill border with black
        );
        cv::imshow("Birds_Eye", birds_image);
        int key = cv::waitKey() & 255;
        if (key == 'u')
            Z += 0.5;
        if (key == 'd')
            Z -= 0.5;
        if (key == 27)
            break;
    }

    vector<cv::Point2f> image_points2;
    vector<cv::Point3f> object_points2;
    for (int i = 0; i < 4; ++i) {
        image_points2.push_back(imgPts[i]);
        object_points2.push_back(cv::Point3f(objPts[i].x, objPts[i].y, 0));
    }
    cv::Mat rvec, tvec, rmat;
    cv::solvePnP(object_points2,    // 3-d points in object coordinate
        image_points2,      // 2-d points in image coordinates
        intrinsic,      // Our camera matrix
        cv::Mat(),      // Since we corrected distortion in the
        // beginning,now we have zero distortion
        // coefficients
        rvec,           // Output rotation *vector*.
        tvec            // Output translation vector.
    );
    cv::Rodrigues(rvec, rmat);

    cout << "rotation matrix: " << rmat << endl;
    cout << "translation vector: " << tvec << endl;
    cout << "homography matrix: " << H << endl;
    cout << "inverted homography matrix: " << H.inv() << endl;

    return 0;
}
```

六、 结论与心得体会

本次实验难度较大，因为是讲了过了一段时间才做的作业，所以对于一些概念了解有些模糊，同时又调用了一些其他的库函数，在理解方面有些困难，但是在做完实验后，我对于相机参数和构建有了更深的认识，收获颇丰。

七、 参考文献

book Ex11-1 (in Chapter 11) , Ex12-1 (in Chapter 12).